A Software Debugging Glossary

Mark Scott Johnson

1981 September 18

(Version 1.0)

Hewlett-Packard Laboratories
Computer Research Center
1501 Page Mill Road
Palo Alto, California 94304

Abstract

This glossary contains 291 definitions of terms dealing with the debugging of computer software. The list includes numerous synonyms, as well as the proper names of debugging systems described in the open literature. Terms and definitions have been obtained from various sources: the software-engineering literature, other software-engineering glossaries, and individual contributions.

Key Words and Phrases:
debugging, glossary, software debugging

CR Categories: 1.4, 4.42

A Software Debugging Glossary

Mark Scott Johnson

1981 September 18

(Version 1.0)

Hewlett-Packard Laboratories
Computer Research Center
1501 Page Mill Road
Palo Alto, California 94304

Abstract

This glossary contains 291 definitions of terms dealing with the debugging of computer software. The list includes numerous synonyms, as well as the proper names of debugging systems described in the open literature. Terms and definitions have been obtained from various sources: the software-engineering literature, other software-engineering glossaries, and individual contributions.

Key Words and Phrases:
debugging, glossary, software debugging

CR Categories: 1.4, 4.42

Preface

This first draft of a software debugging glossary is incomplete in many ways. It is being distributed now so that readers who find it useful in their work can give me feedback before it becomes cast in stone. Comments, suggestions, corrections, and contributions all will be appreciated. Please mail them to me at the following address:

Mark Scott Johnson
Hewlett-Packard Laboratories
Computer Research Center
1501 Page Mill Road, Bldg. 28B
Palo Alto, California 94304

1.0 Introduction

This glossary contains definitions of terms dealing with the debugging of computer software. In cases where the distinctions between hardware, firmware, and software debugging are unclear, such terms are included. Similarly, since the activities of software testing and debugging are closely related, there is some overlap of the corresponding terminologies. For example, terms that categorize bugs (such as "precision error" and "typographic error") are included since they impact debugging methods; whereas terms that categorize testing (such as "functional testing" and "unit testing") are not included since they describe activities that precede debugging. Another area of overlap exists between debugging and static and dynamic program analysis. Terms dealing with the latter are included when they provide significant aids to debugging.

Also included in this glossary are proper names: the names of specific debugging systems and the names of specific compilers and programming environments in which debugging has been emphasized. Only proper names that appear in the open computer-science literature are included, however; inclusion of the numerous home-grown and in-house debuggers is impractical.

The purposes of this glossary are:

- to document current use of software debugging terminology,
- to record the historic development of terminology,
- to provide a useful taxonomy,
- to identify the significant contemporary trends in debugging,
- to present a terminology-based perspective on the debugging literature,
- to define a set of indexing terms for use in future bibliographic systems, and
- to encourage standardization among competing and conflicting terms.

Terms and definitions have been obtained from many sources: the software-engineering literature, other software-engineering glossaries (such as [*Glos79*] and [*Welk79*]), and individual contributions. Quoted and paraphrased definitions are credited by both source and page number; all other definitions either are original to this work or are synthesized from several sources. The list of sources is contained in Section 3. Source citations, as opposed to reference citations, are in italic font.

2.0 Terms and Definitions

Entries in this section are arranged in strict alphabetic order by main term, including single words, compound words, proper names, and acronyms. Each entry follows the customary format for dictionaries:

- The main term is set in large type and ends with a colon.
- Where appropriate, a parenthesized list of abbreviations for the term is given. This list is based on common use.
- Unless an entry consists solely of a cross-reference to another term (for example, "See execution trace."), a part-of-speech label for the term is given, set in italic type. The labels used are *n.* (noun), *pn.* (proper name), *v.* (verb), and *adj.* (adjective). For terms having multiple definitions, the part-of-speech label applies to all definitions unless explicitly overridden.
- Where it can be determined with some degree of reliability, the origin of the term is given, enclosed in brackets. Brackets are also used to explain acronyms.
- Where several definitions for the term are given with the same part-of-speech label, the definitions are numbered and ordered in decreasing order of current or preferred use. Definitions tend to be nested; referent terms are in italic type.
- When current use deviates from generally accepted definitions, a note to this effect is given following the label "Usage:".
- For synonyms, a definition is given once under the term that is the most common or most accurate. The entry for this term also contains a list (labeled by "—SYN.") of all synonyms. Each synonym is given its own entry, which consists merely of a cross-reference to the preferred term.
- If the term has a definition that is similar to, but not synonymous with, another term, a "Compare with" cross-reference is given.

References to further information about particular debugging techniques or systems is given, where appropriate. References are listed in Section 4.

abnormal termination: *n.* exceptional or unexpected termination of a program's execution.

activate (a breakpoint): See *enable* (a breakpoint).

active debugging: *n.* [first used by Scowen [Scow72]] a *debugging action* that requires some user intervention, such as to set breakpoints. Compare with *passive debugging*.

ADB: *pn.* [A (Absolute?) DeBugger] an interactive debugger for assembly-language and C programs, developed in the mid-1970s at Bell Laboratories to run under UNIX [Mara77].

ad-hoc debugging: *n.* [first used by Levine [Levi77]] a debugging style in which the programmer does not attempt to anticipate bugs during the design and coding phases of the program's development. Compare with *planned debugging*.

advice: *n.* [derived from the Interlisp *Advising* subsystem] A *programmable breakpoint*, particularly one set at the beginning or ending of procedures. An advice breakpoint is generally invisible (that is, it is not displayed as a normal user-specified breakpoint).

advise: *v.* [derived from the Interlisp *Advising* subsystem] to *set* an *advice* breakpoint.

Advising: *pn.* [first used in *Interlisp*, but concept originated in SIMSCRIPT [CACI72]] a subsystem of the Interlisp programming system that permits code to be associated with a subroutine without having to understand the internal workings of the subroutine [TeiW78:sec.19]. The code can be placed to execute either before the first statement of the subroutine or after the subroutine exits (but before it returns to its caller). Advising is the means by which procedure-*granularity* breakpoints and traces are implemented in Interlisp. Compare with *BEFORE-AFTER statement*.

AIDS: 1) *pn.* [All-purpose Interactive Debugging System] an interactive debugger for assembly-language and Fortran programs, developed in the late 1960s at New York University for the CDC 6600 [Gris70]. 2) *pn.* [Advanced Interactive Debugging System] a high-level, language-independent, symbolic debugger, developed in the late 1970s at Sperry-Univac Corp. for the Univac Series 1100 [Hart79].

ALADDIN: *pn.* [Assembly Language Assertion-Driven Debugging INterpreter] an assembly-language debugger, developed in the mid-1970s at Colorado State University for the Data General Nova 1200 [Fair79]. Altho its command repertoire was small, it emphasized the use of *assertion-driven breakpoints* over conventional *conditional breakpoints*.

ALCOR: *pn.* a diagnostic compiler for Algol 60, developed in the mid-1960s at the University of Illinois (Urbana-Champaign) for the IBM 7090/7094 [Baye67]. It provided a sophisticated symbolic *postmortem dump*.

Algol-W: *pn.* a diagnostic compiler for an extended subset of Algol 60, developed in the late 1960s at Stanford University [Satt72].

analysis information: *n.* information used to examine the state of program execution, for example, a variable or execution *profile*. —SYN. *program analysis*.

analyzer: *n.* a software tool that provides *analysis information*.

architectural debugging support: *n.* features and attributes of a computer (usually hardware) architecture that aid program debugging.

assembly-language debugging: See *low-level debugging*.

assertion-driven breakpoint: *n.* [first used in *ALADDIN*?] a *breakpoint* that is initiated only if some location-independent predicate evaluates to true. For example, the debugging assertion "WHEN (Foo = 0) BREAK" suspends program execution whenever the value of the variable Foo becomes zero (such as by an assignment or an input statement). Compare with *conditional breakpoint*.

attribute list: *n.* a *static analysis* tool that contains a list of program identifiers, together with their declaration characteristics (such as type and scope) and possibly with information on storage allocation.

automatic debugging: *n.* "the use of automated means to detect inconsistencies between assertions about the inputs and outputs of the various elements of the software." [*Glos79*:7] Usage: Automatic debugging is more appropriately termed "automatic error detection" and considered a testing, rather than a debugging, activity.

backtrace: See *traceback*.

backward execution: See *reversible execution*.

BAIL: *pn.* an interactive debugger for SAIL, developed in the early 1970s at Stanford University to run under TENEX and TOPS-10 [Reis75].

baseline diagram: See *call graph*.

batch debugging: See *noninteractive debugging*.

bebugging: See *bug seeding*.

BEFORE-AFTER statement: *pn.* a construct of the SIMSCRIPT II.5 discrete-event simulation language that permitted a breakpoint to be set immediately before or after the occurrence of a significant event. Compare with *Advising*.

BIDOPS: *pn.* [BI-Directional (O?) Programming System] an interactive programming system for an Algol 60-like language, developed in the late 1970s at the University of New England (Australia) for the DEC 2060 [Hodg80]. Using compiled code, it provided *motion-picture display* and *reversible execution*.

branch execution count: *n.* an *execution profile* limited to statements that (potentially) alter sequential execution flow.

branch trace: *n.* an *execution trace* of successful branches only.

break: 1) *v.* See *interactive break*. 2) *n.* short form of *breakpoint*. 3) *v.* See *initiate* (a breakpoint).

breakpoint: (bkpt, bp, bpt, brkpt) *n.* [first used on the EDSAC [MyeB80:11]] 1) a location in a program's execution at which either some *debugging command* is to be performed or the user wishes to gain control. Breakpoints can be associated either with the access of data values (data breakpoints) or with the execution of program code segments (code breakpoints). The operations that can be performed on breakpoints are *set*, *enable*, *disable*, *initiate*, *display*, and *remove*. Breakpoints can be either *conditional* or *unconditional*, either *dynamic* or *static*, and either *local* or *global*. Usage: Common use implies static, code breakpoints. Data breakpoints are often called *demons*. 2) a time during execution at which either some *debugging command* is to be performed or the user wishes to gain control interactively. 3) a place in a computer program, usually specified by an instruction, where its execution may be interrupted by external intervention or by a monitor program [*Weik79*:8]. 4) when a program is suspended, the particular event that initiated the suspension. 5) the point in a program at which its execution is suspended. 6) *v.* to *set* a breakpoint.

breakpoint table: *n.* a data structure to implement code breakpoints. One method of implementing breakpoints, especially common among low-level debuggers, is to replace one machine instruction of the program being debugged with a call to a subroutine (or an interrupt to a debugging routine) that effects initiation of the breakpoint. The breakpoint table maintains the locations of these code patches and the instructions they replace.

bug: *n.* [first used by Captain Grace M. Hopper in 1951 in reference to the Univac I [Sojk81]] 1) an error in design [*Weik79*:8]. 2) a mistake or malfunction [*Weik79*:8]. 3) any error associated directly or indirectly with a computer, a computer program, information entering or leaving a computer, a computer operator or programmer, or anyone related to or

anything associated with any person engaged in an occupation using computers [*John78a*:70]. —SYN. *fault*.

bug contest: *n.* [first used by Rain [Rain73]] a *debugging technique* in which users of a software system are paid bounties for discovering previously unreported bugs.

bug count: *n.* a record of the number and categories of bugs encountered in a software system over time. It may be used to estimate the software's reliability.

bug seeding: *v.* the process of artificially adding errors to a program with the purpose of obtaining an estimate of the number of natural errors remaining in the program [*Glos79*:11]. —SYN. *bebugging*.

BUGTRAN: *pn.* a noninteractive debugger for Fortran programs, developed in the early 1960s at the University of California (San Diego) [Ferg63].

calibration error: *n.* an error purposely inserted into a program to gauge the completeness of testing to uncover *indigenous errors* [*Glos79*:11].

call graph: *n.* a *static analysis* tool that indicates the relationship among components of a software system, such as what subroutines invoke other subroutines. —SYN. *baseline diagram* and *tier chart*.

call trace: See *subroutine trace*.

cancel (a breakpoint): See *remove* (a breakpoint).

CAPS: *pn.* a diagnostic compiler supporting programs written in Fortran, Cobol, and PL/I, developed in the early 1970s at the University of Illinois (Urbana-Champaign) to run under PLATO IV [Wilt76].

CARTUNE: *pn.* [CObol Analysis Routine for run-time TUNing (E?)] a run-time analysis system for COBOL programs, developed in the mid-1970s at NCR Corp. for the NCR Criterion to run under VRX [Shap78]. The system provided *execution profiles* and *variable profiles*, aided by a firmware implementation of a COBOL-oriented virtual machine.

checkout compiler: See *diagnostic compiler*.

checkpoint: 1) *v.* to save the execution state of a software system, such as a debugger, so that the state can be restored later. 2) *n.* See *data breakpoint*.

clerical error: See *typographic error*.

COBUG: *pn*. [COBol deBUGger] a debugger for COBOL programs, developed in the mid-1970s at NCR Corp. for the NCR Criterion to run under VRX [Shap78]. The system provided selective *execution traces* and selective *variable traces* with paragraph *granularity*, aided by a firmware implementation of a COBOL-oriented virtual machine.

code breakpoint: *n*. a *breakpoint* associated with the execution of some segment of program code. —SYN. *control breakpoint*.

code patch: See *patch*.

code trace: See *execution trace*.

commission: *v*. chiefly British for *debug*. Commission generally refers to the process of debugging a complete computer system, both hardware and software [*Ibbe78*].

comparator: *n*. a software tool that identifies the differences between two versions of the same source program.

computation error: *n*. a deficient implementation of a numeric algorithm.

conditional breakpoint: *n*. 1) a *breakpoint* that is initiated only if some location-dependent predicate evaluates to true. For example, a predicate that yields true when a particular source-code statement has been executed for the n-th time can be used to set a conditional breakpoint, and thus avoid initiation of uninteresting breakpoints. —SYN. *selective breakpoint*. Compare with *assertion-driven breakpoint*. 2) See *assertion-driven breakpoint*. Usage: confusing.

control breakpoint: See *code breakpoint*.

control flow trace: See *execution trace*.

core dump: See *memory dump*.

Cornell Program Synthesizer: *pn*. a programming environment for PL/CS, an extended subset of PL/I, developed in the early 1980s at Cornell University for the DEC PDP-11 and DEC VAX to run under Unix [TeiT81]. Debugging facilities included highlighted *execution traces* with *pacing*, *stepwise execution*, and *motion-picture display*.

cost profile: *n*. an *execution profile* in which the unit of execution frequency is time, either execution or elapsed. —SYN. *timing analysis*.

cross-reference list: *n*. a *static analysis* tool that contains a list of program identifiers, showing where they are declared and used in the program.

cross-referencer: *n*. a software tool that produces a *cross-reference list*.

DAD: *pn*. [Do-All Debugger] a multilingual, multiprocess, and remote debugger, developed in the mid-1970s at SRI Intl. [Vict76a].

data breakpoint: *n*. a *breakpoint* associated with the access of some data value. —SYN. *checkpoint*, *demon*, *storage breakpoint*, and *variable monitoring*.

data flow trace: See *variable trace*.

data sensitivity analysis: *n*. [first used in *ISMS?*] a technique designed to measure *precision errors* which uses a form of *symbolic execution* involving artificial arithmetic to calculate numeric significance at each step of the computation.

data trace: See *variable trace*.

DDS: *pn*. [Dynamic Debugging-Source] an interactive debugger for Coral 662, developed in the early 1970s for the Argus 700 [Pier74].

DDT: 1) *pn*. [Dec Debugging Tape] an assembly-language debugger, developed in the early 1960s at the Massachusetts Institute of Technology for the PDP-1 [Koto61]. It permitted interrogation of machine registers, interpretive execution, *code breakpoints*, *stepwise execution*, *execution traces*, and *patching*. 2) [now: Dynamic Debugging Technique] any of a number of debuggers for the assembly languages of various machines.

deactivate (a breakpoint): See *disable* (a breakpoint).

debug: (dbg) *v*. 1) to isolate and correct a programming *bug*. 2) to examine or test a procedure, routine, or equipment for the purpose of detecting and correcting errors [*Welk79*:19]. 3) *adj*., *n*. short form of *debugging*. 4) *n*. short form of *debugger*. 5) *pn*. a low-level debugger, developed in the early 1960s at Air Force Cambridge Research Laboratories for the Univac M-460 [Evan65]. The system was noteworthy for two reasons: *patches* caused dynamic run-time program relocation as well as automatic updating of the assembly-language source code.

debugger: *n*. a collection of software tools to aid debugging. Typical features include: memory and register initialization; ability to examine, dump, and/or modify memory locations and registers; selectively execute sections of the program by use of breakpoints; and single-step capability [*Welk79*:19]. —SYN. *debugging monitor*, *debugging system*, and *debugging tool*.

debugger generation system: *n*. [idea first proposed by Johnson [John78a:65]] A component of a translator writing system that, given a language definition, produces a language-dependent debugger for programs written in that language. No such system is known to exist.

debugging: (dbg, debug) 1) *n.* the process of isolating and correcting mistakes in computer programs. 2) *adj.* having to do with debugging.

debugging action: *n.* an action that distinguishes debugging from other programming activities. There are two fundamental debugging actions: the ability to manipulate program breakpoints, and the ability to access and possibly modify the execution state of a program (for example, to access values not accessible by the normal scoping rules of the language in which the program is written).

debugging command: *n.* a directive that controls the behavior of a debugger.

debugging language: *n.* a programming or *debugging-command* language geared toward debugging. An example is *Dispel*.

debugging mode: *n.* execution of a program in conjunction with a debugger.

debugging model: *n.* a model that estimates the number of remaining bugs in a software system as a function of time [*Glos79*:31].

debugging monitor: See *debugger*.

debugging routine: *n.* a sequence of either user-specified or library *debugging commands* that can be invoked as a unit, such as at a *breakpoint*.

debugging session: *n.* 1) a period of time during which a debugger is used. 2) a period of time during which any debugging activities occur.

debugging support levels: See *levels of debugging support*.

debugging system: See *debugger*.

debugging technique: *n.* either a manual or an automated method of debugging.

debugging tool: See *debugger*.

debugging variable: *n.* a variable used exclusively for debugging and whose scope is either an entire *debugging session* or a *debugging routine*. Debugging variables are used primarily to maintain information concerning the state of a debugging session.

decompile: *v.* to translate an object program back into a form that resembles (but often is not identical to) the original source program. Decompilation is one way by which an interactive debugger can display a program's suspension point to the user in source-language terms.

deferred action: *n.* [first used in *RAIDE*] the debugging actions of *RAIDE* that are associated with a *programmable breakpoint*.

deferred action list: *n.* [first used in *RAIDE*] the term used in *RAIDE* to describe its *breakpoint table*.

deficiency: *n.* a required function or capability that is missing from a software system [*Glos79*:31].

define (a breakpoint): See *set* (a breakpoint).

delete (a breakpoint): See *remove* (a breakpoint).

demon: See *data breakpoint*.

design error: *n.* a misinterpretation of the specifications of a software system.

diagnostic compiler: *n.* a compiler that emphasizes error detection at compile and run time. Some form of symbolic debugging is often provided as well. —SYN. *checkout compiler* and *student compiler*.

diagnostics: *n.* the output from a software tool (such as a debugger) that indicates and describes a discrepancy [*Glos79*:35].

differential dump: *n.* a *dump* that only displays the changes in state since the last dump was produced. For example, if a main routine Foo calls a procedure Bar and program execution is suspended, a snapshot dump might produce the output:

variables local to procedure Bar:

X = 5 Y = 'HELLO' Z = TRUE

variables local to main routine Foo:

A = 1.56 B = 3.4E-45

If another snapshot dump is produced later during the execution of Bar and only the value of Y has changed in the interim, then a differential snapshot dump might produce the output:

variables of Bar changed since last dump:

Y = 'GOODBYE'

disable (a breakpoint): *v.* to cause a breakpoint that has been *set* to be temporarily inoperative. —SYN. *deactivate*. Compare with *remove*.

Dispel: *pn*. [Debugging SPEcification Language] the debugging command language of *RAIDE*, developed in 1978 at the University of British Columbia for the Amdahl 470 to run under MTS [John78a].

display (a breakpoint): *v*. 1) to indicate what breakpoints have been *set*, if any. 2) to print the *debugging commands* to be executed when some breakpoint is *initiated*.

DITRAN: *pn*. [DIagnostic forTRAN] a noninteractive diagnostic compiler for Fortran, developed in the mid-1960s at the University of Wisconsin for the CDC 1604 [Moul67].

DO-trace: *n*. [first used by Foulk [Foul75]] an *execution trace* with statement-block *granularity*.

dump: [first used on the EDSAC [MyeB80:11]] 1) *n*. a display of some aspect of a program's execution state. 2) *v*. to produce a *dump*.

dump analyzer: *n*. a debugging aid that produces a *symbolic dump*, as opposed to a *memory dump*. —SYN. *dump interpreter*.

dump interpreter: See *dump analyzer*.

DWIM: *pn*. [Do What I Mean] a subsystem of the Interlisp programming system that attempts to automatically correct misspellings, unmatched parentheses, and certain other common errors [TeiW78:sec.17].

DYDE: *pn*. [DYnamic DEbugger] a low-level debugger for assembly-language programs, developed in the mid-1960s at Rand Corp. for the IBM System/360 to run under OS/360 [Jose69].

dynamic analysis: *n*. a tool that incorporates primarily testing-oriented aids to analyze the dynamic execution behavior of a program. Such aids include the *execution profile*, the *cost profile*, the *variable range summary*, and the *variable trace*. Compare with *static analysis*.

dynamic analyzer: *n*. a tool that performs *dynamic analysis* of a program.

dynamic breakpoint: *n*. [first used by Babcicky [Babc81]] a *breakpoint* that is *initiated* based on a dynamic (run-time) characteristic of a program. For example, the dynamic breakpoint "AFTER 20 STATEMENTS BREAK" suspends program execution after twenty more source-level statements are executed. Compare with *static breakpoint*.

dynamic debugging: See *interactive debugging*.

dynamic symbolic evaluation: *n*. a form of *symbolic execution* that maintains a symbolic record during program execution of the path taken for particular input values. Dynamic symbolic evaluation techniques can be used to implement *execution profiles*, *flowback analysis*, and *reversible execution*

[Clar81].

emulate: *v*. to imitate one computer system with another so that the imitating system accepts the same data, executes the same programs, and achieves the same results as the imitated system [*Glos79*:39].

enable (a breakpoint): *v*. to cause a breakpoint that has been *disabled* to become once again operative. —SYN. *activate*.

entomology: *n*. the study of bugs by observation [*VanT74*:154].

epilog breakpoint: *n*. [first used by Johnson [John81d]] a *breakpoint* that *initiates* upon exit from a subroutine or program. —SYN. *postamble breakpoint*. Compare with *prolog breakpoint*.

error: *n*. 1) any discrepancy between a computed, observed, or measured quantity and the true, specified, or theoretically correct value or condition [*Welk79*:26]. 2) an action that results in software containing a *bug*. Classification of errors includes omission or misinterpretation of user requirements in a software specification, *precision errors*, *typographic errors*, *syntactic errors*, *semantic errors*, and *logic errors* [*Welk79*:26].

EXDAMS: *pn*. [EXtendable [sic] Debugging And Monitoring System] an online debugger for PL/I programs, developed in the late 1960s at Rand Corp. [Balz69]. It emphasized the postexecution graphic display of variables, and included *motion-picture display*, *reversible execution*, and *flowback analysis*.

execute (a breakpoint): See *initiate* (a breakpoint).

execution history: *n*. a record of the accesses of data and/or the execution flow of a program. It is needed to implement a *retrospective trace*.

execution in context: *v*. the ability to execute a source-language expression or statement during run time in the context of the suspension point. This ability permits interrogation (and usually modification) of the data state of the suspended program. —SYN. *evaluation in context*.

execution profile: *n*. a *profile* of the execution frequency of code-segments (such as statements or procedures) of a program. The display may include source-code excerpts. The unit of frequency is often a strict numeric count, altho execution or elapsed time can also be used. The latter is termed a *cost profile*. —SYN. *flow profile*.

execution trace: *n*. a display of code-segment labels (such as procedure labels or statement numbers) encountered during program execution. The display may include source-code excerpts; an interactive display may involve highlighting the source code. Some restrictions on the extent of the trace (especially for nested constructs like loops) and the *granularity* of the trace may be provided. For example, a statement-level execution

trace of a subroutine Foo might produce the output:

```
 1: IF (Number <= 0)
      THEN BEGIN
 2:     Number := - Number ;
 3:     Chars[1] := '-'
      END
 7: Index := 0 ;
 8: REPEAT
 9:   Index := Index + 1 ;
10:   Digits[Index] := Number MOD 10 ;
11:   Number := Number MOD 10
12: UNTIL (Number = 0) ;
 8: REPEAT
 9:   Index := Index + 1 ;
10:   Digits[Index] := Number MOD 10 ;
11:   Number := Number MOD 10
12: UNTIL (Number = 0) ;
 8: REPEAT
 9:   Index := Index + 1 ;
10:   Digits[Index] := Number MOD 10 ;
11:   Number := Number MOD 10
12: UNTIL (Number = 0) ;
13: RETURN
```

—SYN. *code trace*, *control flow trace*, *flow trace*, and *forward trace*.

evaluation in context: See *execution in context*.

false breakpoint: *n.* [first used by Johnson [John81d]] a *breakpoint* that is *initiated*, but immediately dismissed. For example, to implement statement-*granularity* code breakpoints on a computer that has architectural support for instruction stepping, the debugger might set false breakpoints on each instruction. The breakpoint handler then checks to see if a statement boundary has been reached. If so, the body of the code breakpoint is executed; otherwise, the breakpoint is dismissed as inapplicable.

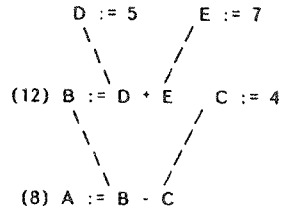
fault: See *bug*.

FID: *pn.* [Fortran Interactive Debugging aid] a high-level debugger for programs written in Fortran, developed in the late 1970s at Los Alamos Scientific Laboratory [Kaze78].

fire (a breakpoint): See *initiate* (a breakpoint).

FLASC: *pn.* a diagnostic compiler for Algol 68, developed in the mid-1970s at the University of Alberta to run under MTS [Thom76].

flowback analysis: *n.* [first used in *EXDAMS*] an inverted tree display of how some variable received a certain value. For example, the following is a flowback showing how the variable A received the value 8.



flow profile: See *execution profile*.

flow trace: See *execution trace*.

foreign debugging: *n.* an in-depth program review conducted by someone other than the implementor to find program errors and improve program reliability. A non-implementor learns the internals of the program to be debugged, constructs appropriate test cases, and debugs just as the implementor would [*Weik79*:31].

forward trace: See *execution trace*. Usage: to distinguish it from a *retrospective trace*.

frequency analyzer: *n.* a software tool that produces an *execution profile*.

GBUG: *pn.* [Graphical deBUGger?] a low-level, online debugger, developed in the mid-1960s at IBM Corp. [Zimm67]. It emphasized *graphic display*.

global breakpoint: *n.* [first used by Henriksen [Henr77]?] a *breakpoint* that remains *enabled* until explicitly *removed*. Compare with *local breakpoint*.

granularity: *n.* the unit of execution (usually in source-language terms) over which a *debugging command* has affect. For example, an execution trace with statement granularity causes trace information to be produced each time a source-level statement boundary is encountered during execution.

graphic debugging: *n.* a debugging style that emphasizes the use of *graphic display*.

graphic display: *n.* a display of debugging information (such as traces and profiles) that takes advantage of a high-resolution output device. A *motion-picture display* is an example.

heap trace: *n.* a *variable trace* restricted to data that has been allocated dynamically in a free-storage area (heap).

HELPER: *pn.* an interactive debugger for assembly-language and Fortran programs, developed in the late 1960s at the Institute for Defense Analyses for the CDC 6600 to run under IDA [Kuls69].

high-level debugging: *n.* debugging in which knowledge of the underlying machine architecture (such as registers) is not required of the user. —SYN. *source-language debugging*. Compare with *low-level debugging*.

hit (a breakpoint): See *initiate* (a breakpoint).

IF: *pn.* [Interactive Fortran] an interactive diagnostic compiler for Fortran, developed in the mid-1970s at the University of British Columbia for the IBM System/360/370 to run under MTS [ORei76].

imperfect debugging: *n.* "an assumption that errors are not always removed or corrected when detected." [*Glos79*:52]

implementation error: *n.* a misinterpretation of a program's specifications.

indigenous error: *n.* an error that has not been purposely introduced. Compare with *calibration error*.

initiate (a breakpoint): *v.* 1) to suspend the activation of a program at some breakpoint so that some *debugging command* can be performed. 2) to automatically execute some predefined *debugging command* associated with a breakpoint. —SYN. *break, execute, fire, hit, invoke, reach, take, trap,* and *trigger*.

INQUIRE: *pn.* the suspended program-state interrogation component of a debugger, developed in the early 1970s at Iowa State University, for the SYMBOL high-level language computer [Ditz80c].

interactive break: *v.* to request a *debugging command* interactively from the user while a program is suspended at a breakpoint. —SYN. *break*.

interactive debugging: *n.* a mode of operation in which the user has direct control of debugging during program execution thru the specification of a *debugging command*. Usage: Interactive debugging implies *online debugging*. —SYN. *dynamic debugging* and *run-time debugging*. Compare with *noninteractive debugging*.

interactive execution: *n.* [first used in *PDE1L*] an *execution trace* in which execution flow is displayed by highlighting the source code of the currently executing statement, often at a slower *pace* than normal execution.

interactive request mode: *n.* [first used in *RAIDE*] the state of *RAIDE* when an *interactive break* occurs.

Interlisp: *pn.* an interpretive programming system for LISP [TeiW78:sec.15]. Debugging facilities includes a sophisticated "Break Package" that supported *code breakpoints* and *execution traces*. Related facilities included *Advising, DWIM, Masterscope,* and the *Programmer's Assistant*.

interprocess breakpoint: *n.* a *breakpoint* that is *set* in one process, but that is to be *initiated* in another process.

interrupt: *n.* temporary suspension of normal program execution to execute a higher priority task [*Welk79*:38].

invoke (a breakpoint): See *initiate* (a breakpoint).

IPE: *pn.* [Incremental Programming Environment] a programming environment designed for Ada and implemented for C, developed in the early 1980s at Carnegie-Mellon University for the DEC VAX to run under Unix [Feil81].

ISMS: *pn.* [Interactive Semantic Modeling System] an experimental program-testing facility for Algol 60, developed in the mid-1970s at Texas A&M University [Fair75]. Thru a combination of *static analysis* and *dynamic analysis* using an execution history database, it included the ability to produce *variable range summaries*, various forms of *execution traces*, *flowback analysis*, and *data sensitivity analysis*.

JDAD: *pn.* [Jovial Do-All Debugger] a variant of *DAD* oriented toward programs written in JOVIAL, developed in the early 1980s at the Rome Air Development Center [Pars79a].

J73AVS: *pn.* [J73 Automated Verification System] a debugging and testing environment for the JOVIAL J73 language, developed in the early 1980s at General Research Corp. [Gann80].

language-dependent debugging: *n. debugging techniques* that are dependent on, or are tailored to, the features of one particular source language. Compare with *language-independent debugging*.

language-independent debugging: *n. debugging techniques* that are independent of any one particular source language. Compare with *language-dependent debugging*.

levels of debugging support: *n.* [first used in *RAIDE*, concept refined by Fairley [Fair80]] the notion that debugging functionality can be ordered by the increasing system support required to implement each function. For example, Fairley identified the following levels of debugging support for an ADA debugging environment [Fair80:20]:

1. diagnostic output statements inserted into the code by the user at compile time,
2. snapshot dumps,
3. selective traces,
4. assertion-driven snapshot dumps and selective traces,
5. interactive break,
6. retrospective trace,
7. dynamic conditional breakpoints,
8. execution-state modifications (such as variable values),
9. simple source-code modifications during execution, and
10. arbitrary source-code modifications during execution.

—SYN. *debugging support levels*.

local breakpoint: *n*. [first used by Henriksen [Henr77]?] a *breakpoint* that is automatically *removed* the first time it is *initiated*. Compare with *global breakpoint*.

logic error: *n*. a deficient implementation of an algorithm.

low-level debugging: *n*. the debugging of programs in which knowledge of the underlying machine architecture (such as registers) is required of the user. —SYN. *assembly-language debugging* and *machine-language debugging*. Compare with *high-level debugging*.

machine-language debugging: See *low-level debugging*.

major error: *n*. an error that potentially can cause a program to reach *abnormal termination*, such as a *logic error* resulting from an infinite loop. Compare with *minor error*.

MANTIS: *pn*. an interactive debugger for Fortran programs, developed in the early 1970s at the University of Oregon for the PDP-10 [Ashb73].

Masterscope: *pn*. a subsystem of the Interlisp programming system that interactively analyzed and cross-referenced user programs. The user queried Masterscope in an English-like notation (for example, "Who calls Foo" and "Show where Foo calls any functions") [TeiW78:sec.20].

memory dump: *n*. a display of memory-location values in their internal form (such as octal or hexadecimal), as opposed to their symbolic form. —SYN. *core dump* and *store dump*. Compare with *symbolic dump*.

MEND: *pn*. [Mdl Executor, aNalyzer, and Debugger] a symbolic debugger for the applicative language MDL, developed in 1977 at the Massachusetts Institute of Technology [Bere78]. Its important features included *motion-picture display*, *graphic display*, its implementation in a separate process from the one being debugged, and its use of an existing editor as the user interface, both for debugging-command specification and for breakpoint setting.

minor error: *n*. an error that does not cause *abnormal termination*, such as a *precision error*. Compare with *major error*.

mistake: *n*. an action producing an unintended result [*Glos79*:69].

motion-picture display: *n*. [first used in *EXDAMS*] a *variable trace* in which the values being traced are displayed graphically during execution. For example, a motion-picture display of the four variables whose values are changed in a procedure Foo might produce the output:

CurrentIndex	NewIndex	OldString	WellDoneFlag
1	5	null	False
	10		
2			
		HI	
			True
15			
		HELLO	

multilingual debugging: *n*. a debugging style that permits the debugging of software in which components have been written in more than one source language.

multimachine debugging: See *remote debugging*.

multiprocess debugging: *n*. a debugging style that permits the debugging of software composed of multiple cooperating processes.

noninteractive debugging: *n*. a mode of operation in which the user has no direct control of debugging during program execution. —SYN. *batch debugging*. Compare with *interactive debugging*.

non-stopping breakpoint: See *programmable breakpoint*.

notify: *n*. [first used in *Smalltalk*] the name of the operation in *Smalltalk* used to implement code breakpoints.

notify window: *n*. [first used in *Smalltalk*] a screen window that appears in *Smalltalk* when the *notify* operation is executed. Most debugging actions are executed within this window.

octal debugging technique: See *ODT*.

ODT: *pn*. [Octal Debugging Technique] a system program, designed to help the user debug programs interactively, in which all addresses, register contents, and memory location contents are expressed in octal notation [*Welk79*:50].

OLDS: *pn.* an online debugger for assembly-language and Fortran programs, developed in the early 1970s at the University of Wisconsin for the Univac 1110 [UWM73].

online debugging: *n.* a mode of operation in which the user invokes a debugger from an online terminal device. Usage: Online debugging does not necessarily imply *interactive debugging*, altho such an implication is common.

pace (of display): *n.* the rate at which output is displayed to the user.

PASES: *pn.* a programming environment for programs written in Pascal, developed in the early 1980s at Yale University [ShaE81]. Debugging facilities included *execution in context* and *stepwise execution*.

passive debugging: *n.* [first used by Scowen [Scow72]] a *debugging action* that requires no user intervention, such as a postmortem dump that is produced automatically upon abnormal termination. Compare with *active debugging*.

patch: 1) *n.* a modification to a program, especially during debugging. Often such a modification takes effect only for a particular activation of the program, and is thus temporary. Usage: Common use implies modification of an object program without concurrent modification of the source program. —SYN. *code patch*. 2) *v.* to perform a *patch*.

PDE1L: *pn.* [Program Development Environment for pl1L] a programming environment for programs written in PL1L, developed in the early 1980s at the IBM T.J. Watson Research Center for the IBM System/370 to run under VM/370/CMS [Mike80]. Debugging facilities included *advice*, *assertion-driven breakpoints*, *data breakpoints*, *execution in context*, *interactive execution*, *retrospective trace*, *stepwise execution*, and *undo*.

PILOT: *pn.* the original name for a subsystem of Interlisp (then called BBN-LISP) containing *DWIM* and a breakpoint package [TeiW69].

place (a breakpoint): See *set* (a breakpoint).

planned debugging: *n.* [first used by Levine [Levi77]] a debugging style in which the programmer designs and codes into the program information specifically to facilitate debugging. Compare with *ad-hoc debugging*.

plant (a breakpoint): See *set* (a breakpoint).

playback: See *reversible execution*.

PL/C: *pn.* an interactive diagnostic compiler for an extended subset of PL/I, developed in the early 1970s at Cornell University for the IBM System/360 to run under OS/360 [Conw77].

PL/CT: *pn.* a display-oriented version of *PL/C*, developed in the mid-1970s at Cornell University to run under CMS [Moor75].

PL/I checkout compiler: *pn.* an interactive diagnostic compiler for PL/I, developed in the late 1960s at IBM Corp. for the IBM System/360 to run under OS/360 [Cuff72].

point of suspension: See *suspension point*.

POODL: *pn.* the virtual machine used to implement a PL/I debugger, developed in the late 1960s at the University of Toronto [Pull69].

postamble breakpoint: See *epilog breakpoint*.

postmortem debugging: *n.* a *debugging technique* in which debugging takes place with a copy of the program state as it existed when the bug was discovered. This technique is useful for situations in which the program must be restored to a consistent state and its execution continued (such as in an airline reservation system or an operating system), where debugging cannot take place in real time.

postmortem dump: *n.* a display of the execution state of a program at its point of *abnormal termination*. Compare with *snapshot dump*.

preamble breakpoint: See *prolog breakpoint*.

precision error: *n.* a calculation that does not achieve a desired accuracy for certain values.

procedure timing: *n.* a *cost profile* with procedure *granularity*.

profile: 1) *n.* a display of the summary activity of some aspect of a program, primarily beneficial for *testing* and optimization purposes. Compare with *execution profile* and *variable profile*. 2) *v.* to produce a *profile*.

profiler: *n.* a software tool that produces a *profile*.

program analysis: See *analysis information*.

programmable breakpoint: *n.* a *breakpoint* that automatically invokes some previously specified *debugging command* when *initiated*, without making a request interactively to the user. —SYN. *advice* and *non-stopping breakpoint*.

programmer's apprentice: *n.* a collection of software tools to aid programers in the design, implementation, and maintenance of their programs. A debugger is an essential component of any programmer's apprentice.

Programmer's Assistant: *pn*. a subsystem of the Interlisp programming system that maintained a command-execution history and supported a facility to *undo* commands [TeiW78:sec.22].

prolog breakpoint: *n*. [first used by Johnson [John81d]] a *breakpoint* that *initiates* upon entry of a subroutine or program. —SYN. *preamble breakpoint*. Compare with *epilog breakpoint*.

RAIDE: *pn*. [Run-time Analysis and Interactive Debugging Environment] a high-level, language-independent, symbolic debugger, developed in 1978 at the University of British Columbia for the Amdahl 470 to run under MTS [John78a].

reach (a breakpoint): See *initiate* (a breakpoint).

real-time debugging: *n*. debugging of software that interacts directly with time-critical events, generally associated with physical devices. Real-time debugging is characterized by the general non-repeatability of the programming errors discovered.

remote debugging: *n*. a mode of operation in which the user and the program being debugged are physically separated, such as in a computer network or a host/target environment. —SYN. *multimachine debugging*.

remove (a breakpoint): *v*. to permanently reverse the affect of setting a breakpoint. Removing a breakpoint is generally more severe that disabling one since setting a breakpoint is generally more difficult than enabling one. —SYN. *cancel* and *delete*. Compare with *disable*.

replay: See *reversible execution*.

resident debugger: *n*. a debugger that resides completely within the main memory of a computer, at least while actual debugging is being done.

retrace: See *retrospective trace*.

retrospective trace: *n*. an historic display of the execution path of a program. A retrospective trace differs from a *trace* in that the latter is produced cumulatively during execution, whereas the former is produced on request. A retrospective trace is implemented by maintaining an *execution history* of the program. For example, a retrospective trace of the last five procedure invocations might produce the output:

```
Foo called from statement 38 in Bar.
Bar called from statement 120 in Bell.
Bell called from statement 5 in Main.
Blah called from statement 4 in Main.
Init called from statement 1 in Main.
```


A retrospective trace differs from a *traceback* in that the latter only displays currently active program units, whereas the former simply displays a summary of execution. Usage: Altho applicable to both code and data, common use implies retrospective tracing of code-segments. —SYN. *retrace*.

reverse execution: See *reversible execution*.

reversible execution: *n*. [first used by Zelkowitz [Zelk73]?] a *debugging technique* in which a history of program execution is maintained and then displayed under the user's control, in either the forward or the reverse direction, and often selectively and at higher speed than the original execution display. —SYN. *backward execution*, *playback*, *replay*, and *reverse execution*.

run-time check: *n*. code that is compiled into a program to detect a programming-language restriction that cannot be enforced at compile time (for example, uninitialized variable, value range, and null pointer referencing). Run-time checks help detect *logic errors*. —SYN. *run-time diagnostics*.

run-time debugging: See *interactive debugging*.

run-time diagnostics: See *run-time check*.

SDB: *pn*. [Symbolic DeBugger] a symbolic debugger for C programs, developed in the late 1970s at Bell Laboratories to run under UNIX [Kats79].

SDS: *pn*. [Symbolic Debugging System] an interactive debugger for assembly-language, Fortran, PL/I, and PL360 programs, developed in the early 1970s at the University of British Columbia for the IBM System/360/370 to run under MTS [Ball77].

selective breakpoint: See *conditional breakpoint*.

selective dump: *n*. a *dump* that involves only selected aspects of a program's execution.

selective trace: *n*. 1) a *variable trace* that involves only selected variables. 2) a *variable trace* that is limited in scope. For example, a trace of the value changes for some variable might be limited to the first (or last) "n" changes.

semantic error: *n*. a misunderstanding in the use of some construct in the source language.

set (a breakpoint): *v.* to describe where a breakpoint should exist in a program. —SYN. *define*, *place*, and *plant*.

SIMDDT: *pn.* [SIMula Dynamic Debugging Technique] an interactive debugger for SIMULA programs, developed in the mid-1970s at the Swedish National Defense Research Institute for the DECsystem-10 [Palm77].

SIMDEB: *pn.* [SIMula DEBugger] an interactive debugger for SIMULA programs, developed in the late 1970s for the Data General Eclipse [Holm81].

SIMDEBUG: *pn.* [SIMula DEBUGger] an interactive debugger for SIMULA programs, developed in the late 1970s for the IBM System/360/370 [Babc81].

Simon: *pn.* a project-management aid that retains a history by category of all programming bugs, developed in the early 1970s at Mitre Corp. [Clap74].

single-step execution: See *stepwise execution*.

Smalltalk: *pn.* a object-oriented programming language and environment, developed in the 1970s at the Xerox Palo Alto Research Center for the Alto [Tesl81]. The system emphasized the use of a high-resolution display and highly interactive execution. Debugging facilities included a *notify* operation, *execution in context*, *stepwise execution*, *traceback*, and an *undo* operation.

snap: 1) *n.* short form of *snapshot dump*. 2) *v.* to produce a *snapshot dump*.

snapshot: 1) *n.* short form of *snapshot dump*. 2) *v.* to produce a *snapshot dump*.

snapshot dump: *n.* a display of the execution state of a program at a specified point or time. The display usually includes the values of all variables accessible at the suspension point. For example, if a main routine Foo calls a procedure Bar and program execution is suspended, a snapshot dump might produce the output:

variables local to procedure Bar:

X = 5 Y = 'HELLO' Z = TRUE

variables local to main routine Foo:

A = 1.56 B = 3.4E-45

A snapshot dump can take the form of a *differential dump*. Compare with *postmortem dump*.

source-language debugging: See *high-level debugging*.

SPAM: *pn.* [Specialized Prodebugging Abstract Machine] the virtual machine used to implement *RAIDE*, developed in 1978 at the University of British Columbia for the Amdahl 470 to run under MTS [John78a].

SPLINTER: *pn.* [Scientific PL/I INTERpreter] a noninteractive diagnostic compiler for a subset of PL/I, developed in the mid-1960s at Boeing Co. for the Univac 1108 to run under EXEC II [Glas68].

statement count: See *statement execution count*.

statement execution count: *n.* an *execution profile* with statement *granularity*. —SYN. *statement count*.

static analysis: *n.* analysis of program characteristics based solely on its source code. Compare with *dynamic analysis*.

static analyzer: *n.* a tool that performs *static analysis* of a program.

static breakpoint: *n.* [first used by Babcicky [Babc81]] a *breakpoint* that is *initiated* based on a static (compile-time) characteristic of a program. For example, the static breakpoint "AFTER Foo ENTRY BREAK" suspends program execution immediately after the procedure Foo is entered. Compare with *dynamic breakpoint*.

step (thru execution): See *stepwise execution*.

stepwise execution: *v.* the ability to dynamically step thru program execution, stopping periodically (for example, at source-level statement boundaries) so the user can interrogate the program's state. —SYN. *single-step execution* and *step*.

storage breakpoint: See *data breakpoint*.

store dump: *n.* chiefly British for *memory dump*.

student compiler: See *diagnostic compiler*.

SWARD: *pn.* [SoftWAre Reliability-Directed machine] an architectural design that supported reliable software and debugging, developed in 1977 at the Polytechnic Institute of New York [MyeG78].

subroutine trace: *n.* an *execution trace* of the subroutines invoked during program execution. The trace may be for all or only selected subroutines, and it may include the values of the parameters passed to the subroutines and the values returned by them. —SYN. *call trace*.

suspension point: *n.* the location at which a suspended subroutine activation will resume execution. Usage: Altho any given suspended program normally has many suspension points, common use often refers specifically to the most recently suspended subroutine activation, the one that invoked some debugging action. —SYN. *point of suspension*.

symbolic debugging: *n.* the debugging of programs in terms of their source-level names and constructs. Usage: Symbolic debugging often implies *high-level debugging*.

symbolic dump: *n.* a display of a data value in a form appropriate to the data's type. —SYN. *variable dump*. Compare with *memory dump*.

symbolic execution: *n.* A *static analysis* technique in which selected paths of a program are analyzed, producing as output a symbolic history of the conditions that distinguish each path and the symbolic values of all variables along each path.

symbolic trace: *n.* a linear listing of source statements and branch predicates that occur along an execution path in a program [*Huan80*]. For example, a symbolic trace of the following program:

```
proc Hanoi (N : Integer ; Me,De,Ma : Char) ;
  if (N > 0) then
    Hanoi(N-1,Me,Ma,De) ;
    Print(N,Me,Ma) ;
    Hanoi(N-1,De,Me,Ma)
  fi
```

might produce the following output for Hanoi(2,"A","B","C"):

```
proc Hanoi (N(2) ; Me("A"),De("B"),Ma("C")) ;
((N > 0) is True)
Hanoi(N-1,Me,Ma,De) ;
  proc Hanoi (N(1) ; Me("A"),De("C"),Ma("B")) ;
  ((N > 0) is True)
  Hanoi(N-1,Me,Ma,De) ;
    proc Hanoi (N(0) ; Me("A"),De("B"),Ma("C")) ;
    ((N > 0) is False)
  Print(N(1),Me("A"),Ma("B")) ;
  Hanoi(N-1,De,Me,Ma) ;
    proc Hanoi (N(0) ; Me("C"),De("A"),Ma("B")) ;
    ((N > 0) is False)
Print(N(2),Me("A"),Ma("C")) ;
Hanoi(N-1,De,Me,Ma) ;
  proc Hanoi (N(1) ; Me("B"),De("A"),Ma("C")) ;
  ((N > 0) is True)
  Hanoi(N-1,Me,Ma,De) ;
```



```
    proc Hanoi (N(0) ; Me("B"),De("C"),Ma("A")) ;
    ((N > 0) is False)
  Print(N(1),Me("B"),Ma("C")) ;
  Hanoi(N-1,De,Me,Ma) ;
    proc Hanoi (N(0) ; Me("A"),De("B"),Ma("C")) ;
    ((N > 0) is False)
```

syntactic error: *n.* a violation of the grammatical rules defining a programming language.

take (a breakpoint): See *initiate* (a breakpoint).

TALK: *pn.* [Take A LooK] a debugger for CS-1, developed in early 1960s at Sperry-Univac Corp. [VerS64].

testing: *n.* the process of verifying that a computer program behaves according to its specifications.

tier chart: See *call graph*.

timing analysis: See *cost profile*.

timing analyzer: *n.* a software tools that produces a *cost profile*.

Tinker: *pn.* an experimental interactive programming environment for LISP programs, developed in the early 1980s at the Massachusetts Institute of Technology [Lieb80]. It integrated program design, testing, and debugging.

trace: [first used on the EDSAC [MyeB80:11]] 1) *n.* a display of the dynamic activity of some aspect of a program. Types of traces include the *execution trace*, the *retrospective trace*, the *subroutine trace*, and the *variable trace*. Traces can be either complete for an entire program or selective within a portion of a program. 2) *v.* to produce a *trace*.

traceback: *n.* 1) a display of the current invocation state of a program at a point of suspension. The display may include the values of the actual parameters of the suspended procedures. For example, if a main routine Foo calls a procedure Bar, which in turn calls a procedure Bell, and program execution is suspended, a traceback at the suspension point might produce the output:

```
Program suspended at statement 4 in Bell.
Bell called from statement 47 in Bar.
Bar called from statement 150 in Foo.
```

—SYN. *backtrace*. Compare with *retrospective trace*. 2) See *retrospective trace*. Usage: confusing.

transient deferred action: *n.* [first used in *RAIDE*] a *programmable breakpoint* in *RAIDE* that cancels itself the first time it is *initiated*.

trap: *n.* 1) a CPU-initiated *interrupt* automatically generated when a predetermined condition (such as an illegal instruction or a breakpoint) is detected [*Welk79*:75]. 2) See *breakpoint*. Usage: confusing. 3) *v.* See *initiate* (a breakpoint).

trigger (a breakpoint): See *initiate* (a breakpoint). Usage: particularly with regard to hardware breakpoints.

typographic error: *n.* a mistake made in entering a program into a computer. —SYN. *clerical error*.

unconditional breakpoint: *n.* a *breakpoint* that is not conditional. Examples of unconditional breakpoints include when some statement in a procedure is about to be executed and when some variable's value is about to be changed. A breakpoint is *initiated* whenever the executing program reaching one of these states.

undo: *v.* to reverse the affect of some previously executed action, such as a *debugging command*.

UW-Pascal: *pn.* a diagnostic compiler for Pascal, developed in the late 1970s at the University of Wisconsin [Fisc80]. It emphasized *run-time checking*, particularly of pointers, reference parameters, and variant records.

validation: *n.* "the process of determining the level of conformance between an operational software system and its specifications" [*Ridd80*:213].

value trace: See *variable trace*.

variable dump: See *symbolic dump*.

variable monitoring: See *data breakpoint*.

variable profile: *n.* a *profile* of the number of accesses of, or changes to, the variables in a program. The profile may be for all or only selected variables.

variable range summary: *n.* [first used in *ISMS?*] a *dynamic analysis* tool which displays after execution the range of values that variables received during execution. The display may be for all or only selected variables, and it may be for only one program execution or accumulated over a series of executions.

variable trace: *n.* a display of the names and values of variables accessed or changed during program execution. The display may include source-code excerpts; an interactive display may involve highlighting the source code and/or displaying variable values in a window. The latter is termed a

motion-picture display. For example, a trace of the value changes for the variable Index in procedure Foo might produce the output:

```
Index  #: Statement

  0    7: Index := 0 ;
  1    9:   Index := Index + 1 ;
  2    9:   Index := Index + 1 ;
  3    9:   Index := Index + 1 ;
```

—SYN. *data flow trace*, *data trace*, and *value trace*. Compare with *selective trace*.

virtual debugging machine: *n.* a virtual machine designed specifically to support debugging. Examples include *POODL* and *SPAM*.

3.0 Sources

[Glos79] Gloss-Soler, Shirley A. (ed.) The DACS Glossary. Data & Analysis Center for Software (1979 October), 147pp.

[Huan80] Huang, J.C. Instrumenting Programs for Symbolic-Trace Generation. *Computer* 13:12 (1980 December), 17-23.

[Ibbe78] Ibbett, R.N. and Capon, P.C. The Development of the MU5 Computer System. *Comm. of the ACM* 21:1 (1978 January), 13.

[John78a] Johnson, Mark Scott. *The Design and Implementation of a Run-Time Analysis and Interactive Debugging Environment*. Technical Report 78-6, Ph.D. Thesis: Univ. of British Columbia (1978 August), 149pp.

[Ridd80] Riddle, William E. and Fairley, Richard E. (eds.) *Software Development Tools*. Springer-Verlag (1980), 280pp.

[VanT74] Van Tassel, Dennie L. *Program Style, Design, Efficiency, Debugging, and Testing*. Prentice-Hall (1974), 117-165.

[Weik79] Weik, Martin H. (ed.) Computer Dictionary (Draft Standard). IEEE Computer Society (1979), 79pp.

4.0 References

[Ashb73] Ashby, Gordon; Salmonson, Loren; and Heilman, Robert. Design of an Interactive Debugger for FORTRAN: MANTIS. *Software—Practice and Experience* 3:1 (1973 January-March), 65-74.

[Babc81] Babcicky, Karel. SIMDEBUG—An Interactive Debugger for the IBM SIMULA. *SIMULA Newsletter* 9:2 (1981 May), 14-17.

[Ball77] Ballard, Alan John. UBC DEBUG: The Symbolic Debugging System. Univ. of British Columbia (1977 September), 126pp.

[Balz69] Balzer, Robert M. EXDAMS—EXtendable [sic] Debugging and Monitoring System. *Proc. AFIPS Conf.* 34 (1969), 567-580.

[Baye67] Bayer, Rudolf; Gries, David; Paul, M.; and Wiehle, H.R. The ALCOR Illinois 7090/7094 Post Mortem Dump. *Comm. of the ACM* 10:12 (1967 December), 804-808.

[Bere78] Berez, Joel Mayer. A Dynamic Debugging System for MDL. Technical Report MIT/LCS/TM-94, B.S. Thesis: Massachusetts Institute of Technology (1978 January), 53pp.

[CACI72] Consolidated Analysis Centers, Inc. *SIMSCRIPT II.5 Reference Handbook*. (1972), ??pp.

[Clap74] Clapp, J.A. and Sullivan, J.E. Automated Monitoring of Software Quality. *Proc. AFIPS Conf.* 43 (1974), 337-341.

[Clar81] Clarke, Lori A. and Richardson, Debra J. Symbolic Evaluation Methods for Program Analysis. *Program Flow Analysis: Theory and Applications* Muchnick, Steven S. and Jones, Neil D. (eds.) Prentice-Hall (1981), 264-300.

[Conw77] Conway, Richard Walter; Moore, Charles, Jr.; and Worona, Steven L. An Interactive Version of the PL/C Compiler. *Proc. ACM Annual Conf.* (1977 October), 308-314.

[Cuff72] Cuff, R.N. A Conversational Compiler for Full PL/I. *Computer Journal* 15:2 (1972 May), 99-104.

[Ditz80c] Ditzel, David R. High Level Language Debugging Tools on the SYMBOL Computer System. *Proc. 1980 Workshop on High-Level Language Computer Architecture* (1980 May), 247-255.

[Evan65] Evans, Thomas G. and Darley, D. Lucille. DEBUG—An Extension to Current Online Debugging Techniques. *Comm. of the ACM* 8:5 (1965 May), 321-326.

[Fair75] Fairley, Richard E. An Experimental Program-Testing Facility. *IEEE Trans. on Software Engineering* SE-1:4 (1975 December), 350-357.

[Fair79] Fairley, Richard E. ALADDIN: Assembly Language Assertion Driven Debugging INterpreter. *IEEE Trans. on Software Engineering* SE-5:4 (1979 July), 426-428.

[Fair80] Fairley, Richard E. Ada Debugging and Testing Support Environments. *SIGPLAN Notices* 15:11 (1980 November), 16-25.

[Feil81] Feiler, Peter H. and Medina-Mora, Raul. An Incremental Programming Environment. *Proc. Fifth Intl. Conf. on Software Engineering* (1981 March 9-12), 44-53.

[Ferg63] Ferguson, H. Earl and Berner, Elizabeth. Debugging Systems at the Source Language Level. *Comm. of the ACM* 6:8 (1963 August), 430-432.

[Fisc80] Fischer, Charles N. and LeBlanc, Richard J. The Implementation of Run-Time Diagnostics in Pascal. *Trans. on Software Engineering* SE-6:4 (1980 July), 313-319.

[Foul75] Foulk, Clinton R. The DO Trace: A Simple and Effective Method for Debugging GOTO-Free Programs. *SIGPLAN Notices* 10:9 (1975 September), 11-18.

[Gann80] Gannon, C. A Debugging, Testing, and Documentation Tools for JOVIAL J73. *Proc. COMPSAC 80* (1980 October 27-31), 634-639.

[Glas68] Glass, Robert L. SPLINTER—A PL/I Interpreter Emphasising Debugging Capability. *Computer Bulletin* 12:5 (1968 September), 180-185.

[Gris70] Grishman, Ralph. The Debugging System AIDS. *Proc. AFIPS Conf.* 36 (1970), 59-64.

[Hart79] Hart, Jolene J. The Advanced Interactive Debugging System. *SIGPLAN Notices* 14:12 (1979 December), 110-121.

[Henr77] Henriksen, James O. An Interactive Debugging Facility for GPSS. *Proc. IEEE 1977 Winter Simulation Conf.* (1977 December 5-7), 331-338.

[Hodg80] Hodgson, L.I. and Porter, M. BIDOPS: A Bi-Directional Programming System. *Australian Computer Science Comm.* 2:3 (1980 June 80), 349-355.

[Holm81] Holm, Per and Magnusson, Boris. SIMDEB—An Interactive Debugger for the Data General Eclipse SIMULA. *SIMULA Newsletter* 9:2 (1981 May), 9-13.

[John78a] Johnson, Mark Scott. *The Design and Implementation of a Run-Time Analysis and Interactive Debugging Environment*. Technical Report 78-6, Ph.D. Thesis: Univ. of British Columbia (1978 August), 149pp.

[John81d] Johnson, Mark Scott. Some Requirements for Architectural Support of Software Debugging. Hewlett-Packard Laboratories (1981 August), 15pp.

[Jose69] Josephs, William H. An On-Line Machine Language Debugger for OS/360. *Proc. AFIPS Conf.* 35 (1969), 179-186.

[Kats79] Katseff, Howard P. SDB: A Symbolic Debugger. *Unix Programmer's Manual, 7th. Ed., Vol. 2C* Bell Laboratories (1979 January), 7pp.

[Kaze78] Kazek, Chester S., Jr. FORTRAN Extended Interactive Debugging Aid. Report UC-32, Los Alamos Scientific Laboratory (1978 September), 11pp.

[Koto61] Kotok, A. DEC Debugging Tape. Technical Memo MIT-1, Massachusetts Institute of Technology (1961 December), ??pp.

[Kuls69] Kulsrud, Helene E. HELPER: An Interactive Extensible Debugging System. *Proc. Second Symp. on Operating Systems Principles* (1969 October), 105-111.

[Levi77] Levine, Lawrence H. Debugging, Planned or Ad Hoc, Which is More Effective? *Journal of Educational Data Processing* 14:4 (1977), 1-9.

[Lieb80] Lieberman, Henry and Hewitt, Carl. A Session with TINKER: Interleaving Program Testing with Program Design. *Proc. of the Lisp Conf.* (1980 August), 90-99.

[Mara77] Maranzano, J.F. and Bourne, S.R. A Tutorial Introduction to ADB. *Unix Programmer's Manual, 7th. Ed., Vol. 2A* Bell Laboratories (1979 January), 27pp.

[Mike80] Mickelsons, M. and Wegman, Mark N. PDE1L: The PL1L Program Development Environment Princples of Operation. Technical Report RC-8513, IBM T.J. Watson Research Center (1980 October), 61pp.

[Moor75] Moore, C.G., III; Worona, Steven L.; and Conway, Richard Walter. PL/CT—A Terminal Version of PL/C. Technical Report 75-259, Cornell Univ. (1975 September), 7pp.

[Moul67] Moulton, P.G. and Muller, M.E. DITRAN—A Compiler Emphasizing Diagnostics. *Comm. of the ACM* 10:1 (1967 January), 45-52.

[MyeB80] Myers, Brad Allan. Displaying Data Structures for Interactive Debugging. Xerox Palo Alto Research Center (1980 June), 97pp.

[MyeG78] Myers, Glenford J. *Advances In Computer Architecture*. John Wiley & Sons (1978), 314pp.

[ORei76] O'Reilly, Dennis. UBC IF: The Interactive FORTRAN Manual. Univ. of British Columbia (1976 September), 121pp.

[Palm77] Palme, Jacob. SIMDDT—For Conversational Debugging of SIMULA Programs. *SIMULA Newsletter* 5:2 (1977 May), 13-16.

[Pars79a] Parsley, Bruce L.; Lehtman, Harvey G.; and Kahn, Susan. On-Line Programmer's Management System. Report RADC-TR-79-205, Rome Air Development Center (1979 August), 66pp.

[Pier74] Pierce, R.H. Source Language Debugging on a Small Computer. *Computer Journal* 17:4 (1974 November), 313-317.

[Pull69] Pullam, John M. An Object-time Diagnostic Facility for High-level Languages. M.S. Thesis: Univ. of Toronto (1969), 155pp.

[Rain73] Rain, Mark. Two Unusual Methods for Debugging System Software. *Software—Practice and Experience* 3:1 (1973 January-March), 61-63.

[Reis75] Reiser, John F. BAIL—A Debugger for SAIL. Technical Report STAN-CS-75-523, Stanford Univ. (1975 October), 24pp.

[Satt72] Satterthwaite, Edwin Hallowell, Jr. Debugging Tools for High Level Languages. *Software—Practice and Experience* 2:3 (1972 July-September), 197-217.

[Scow72] Scowen, R.S. Debugging Computer Programs—A Survey with Special Emphasis on ALGOL. Report NAC-21, National Physical Laboratory (1972 June), 36pp.

[ShaE81] Shapiro, Ehud; Collins, Gregg; Johnson, Lewis; and Ruttenberg, John. PASES: A Programming Environment for PASCAL. *SIGPLAN Notices* 16:8 (1981 August), 50-57.

[Shap78] Shapiro, Michael D. The Criterion COBOL System. *Proc. AFIPS Conf.* (1978), 1049-1054.

[Sojk81] Sojka, Deborah and Dorn, Philip H. Magic Moments in Software. *Datamation* 27:9 (1981 August 25), 8.

[TeiT81] Teitelbaum, Tim; Reps, Thomas; and Horwitz, Susan. The Why and Wherefore of the Cornell Program Synthesizer. *SIGPLAN Notices* 16:6 (1981 June), 8-16.

[TeiW69] Teitelman, Warren. Toward a Programming Laboratory. *Proc. Intl. Joint Conf. on Artificial Intelligence* (1969 May), 1-8a.

[TeiW78] Teitelman, Warren. Interlisp Reference Manual. Xerox Palo Alto Research Center (1978 October), 717pp.

[Tesl81] Tesler, Larry. The Smalltalk Environment. *Byte* 6:8 (1981 August), 90-147 (intermittent).

[Thom76] Thomson, C.M. Error Checking, Tracing, and Dumping in an ALGOL 68 Checkout Compiler. *Proc. Fourth Intl. Conf. on the Design and Implementation of Algorithmic Languages* (1976 June), 93-98.

[UWM73] University of Wisconsin. OLDS Reference Manual for the 1110. Univ. of Wisconsin, Madison (1973 February), 20pp.

[VerS64] ver Steeg, R.L. TALK—A High-Level Source Language Debugging Technique with Real-Time Data Extraction. *Comm. of the ACM* 7:7 (1964 July), 418-419.

[Vict76a] Victor, Kenneth E. The Design and Implementation of DAD, a Multi-Process, Multi-Machine, Multi-Language, Interactive Debugger. SRI Intl. (1976 August), 51pp.

[WilT76] Wilcox, Thomas R.; Davis, Alan Mark; and Tindall, Michael H. The Design and Implementation of a Table Driven, Interactive Diagnostic Programming System. *Comm. of the ACM* 19:11 (1976 November), 609-616.

[Zelk73] Zelkowitz, Marvin V. Reversible Execution. *Comm. of the ACM* 16:9 (1973 September), 566.

[Zimm67] Zimmerman, Luther L. On-Line Program Debugging—A Graphic Approach. *Computers and Automation* 16:10 (1967 November), 30-31,34.